# Syntax Parser -

That parse your code and convert into the computer understandable language .

# Lexical Environment:

Where the function and variable sits physically .

Function abc(){

Function inner(){

}

}

# Execution context-

Manage the code that has been currently running .

Java Script object:

Name value pair

Var a = {

b : function(){}

C :

{

Xyz : local

Yze : ity

}

}

# Execution Context (Global):

The base execution context is global execution context.

Global Execution Context creates two things below

This

(Window)

Global Object

window

New browser tab means new global (window) execution context.

Global Means – “Not Inside a Function”. At a base level in browser global object and this will be equal

Outer environment will be null in case of base execution context.

# Outer Environment:

Outer environment depends on the Lexical position. “Lexical Position means where the code actually sits.”

## Hoisting

Setting up the memory space for the variable and function called the hoisting.

### Execution context behaviour phase:

#### Creation phase:

Global Object, this, Outer Environment

In creation phase it sets up memory space for variable and function and not referenced to actual value.

All variable in java script set to undefined in creation phase.

#### Execution phase:

Runs the code line by line (including the assignment operator).

# Question:- (This logic)I have this Javascript constructor-

function TestEngine() {

this.id='Foo';

}

TestEngine.prototype.fooBar = function() {

this.id='bar';

return true;

}

TestEngine.prototype.start = function() {

this.fooBar();

}

TestEngine.prototype.startMethod = function() {

inter = setInterval(this.start, 200);

}

var test = new TestEngine();

test.startMethod();

Gives me this error -

Uncaught TypeError: Object [object global] has no method 'fooBar'

I tried console.log and found out that when I call this.start from within setInterval, this points to the window object. Why is this so?

## Answer

The this pointer can point to one of many things depending upon the context:

1. In constructor functions (function calls preceded by new) this points to the newly created instance of the constructor.
2. When a function is called as a method of an object (e.g. obj.funct()) then the this pointer inside the function points to the object.
3. You can explicitly set what this points to by using call, apply or bind.
4. If none of the above then the this pointer points to the global object by default. In browsers this is the window object.

In your case you're calling this.start inside setInterval. Now consider this dummy implementation of setInterval:

function setInterval(funct, delay) {

// native code

}

It's important to understand that start is not being called as this.start. It's being called as funct. It's like doing something like this:

var funct = this.start;

funct();

Now both these functions would normally execute the same, but there's one tiny problem - the this pointer points to the global object in the second case while it points to the current this in the first.

An important distinction to make is that we're talking about the this pointer inside start. Consider:

this.start(); // this inside start points to this

var funct = this.start;

funct(); // this inside funct (start) point to window

This is not a bug. This is the way JavaScript works. When you call a function as a method of an object (see my second point above) the this pointer inside the function points to that object.

In the second case since funct is not being called as a method of an object the fourth rule is applied by default. Hence this points to window.

You can solve this problem by binding start to the current this pointer and then passing it to setInterval as follows:

setInterval(this.start.bind(this), 200);

That's it. Hope this explanation helped you understand a little bit more about the awesomeness of JavaScript.

# Arguments function

(a b c){

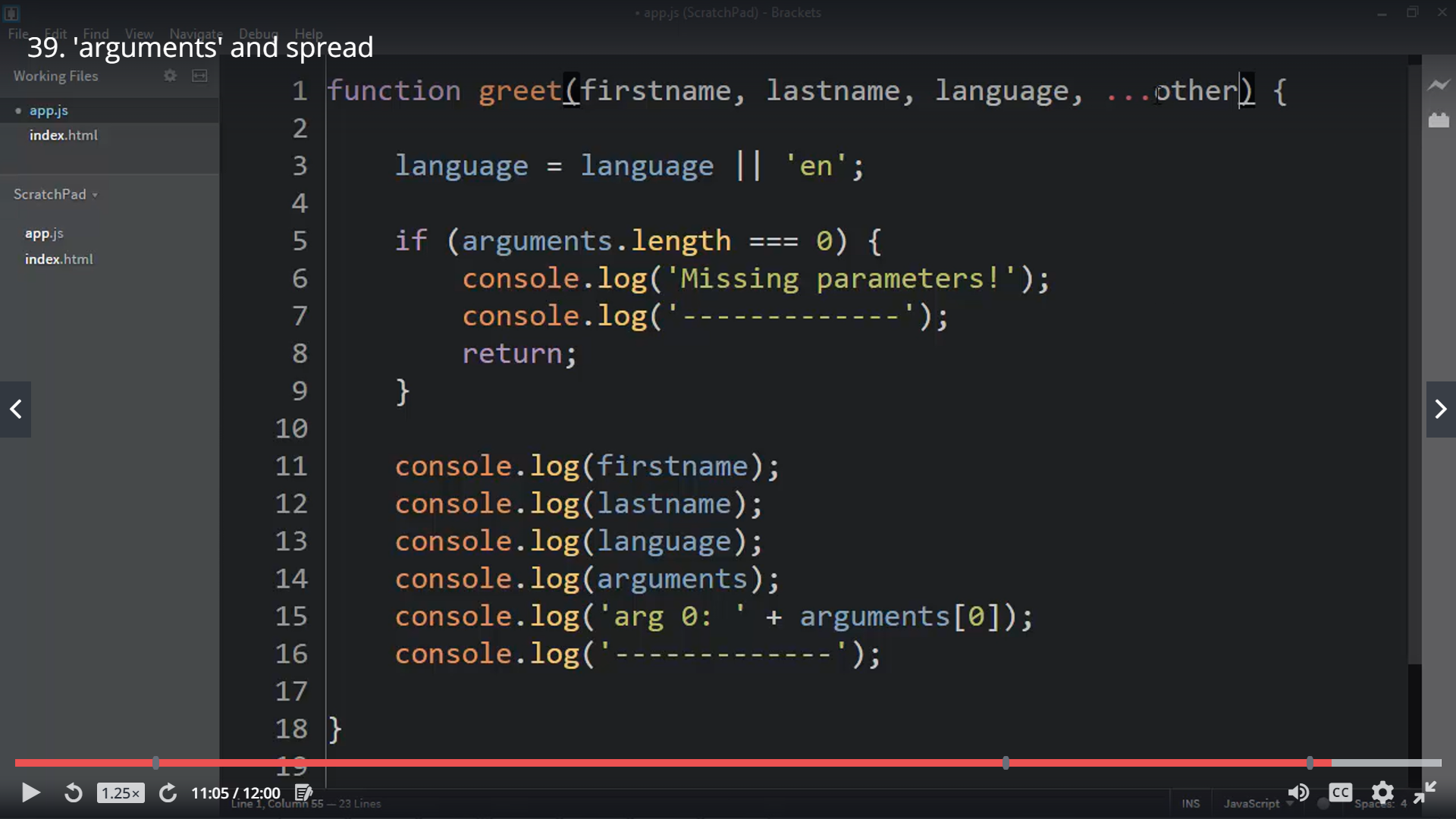
Console.log(arguments);

}

(abc) in case of all arguments prints [a,b,c]

() in case of no arguments prints []

Arguments key word is used to validate the function parameters.



# Function overloading

Create the wrapper over function with different name same name function with different arguments not allowed

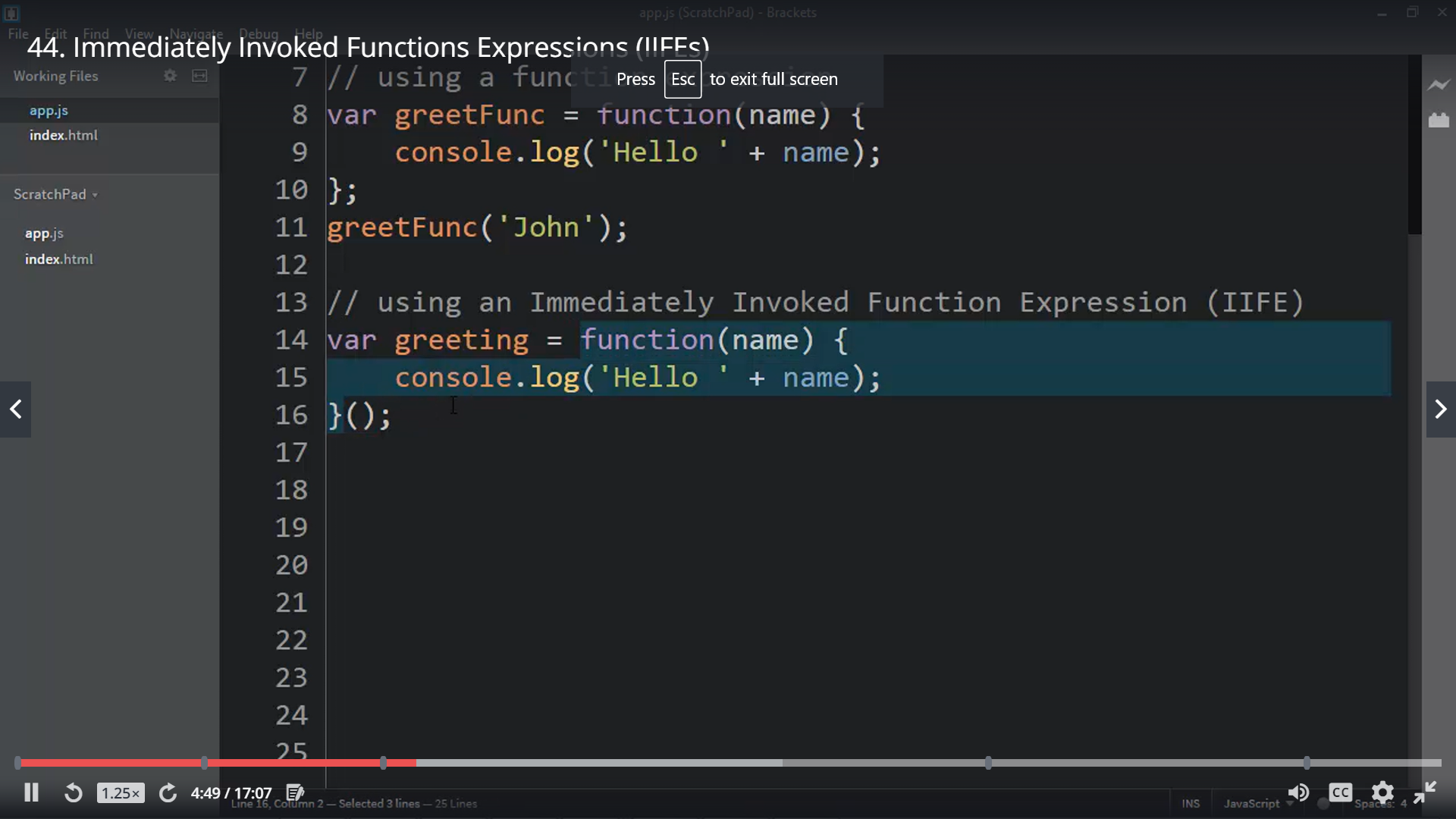


# Automatic ; insertion

Automatic ; insertion and will not be able return the object because Java script compiler checks and sends it



# Immediately Invoked Functions



Function as a expression (you can write in this format only)

